Clase 4 — 11.11.25

#IntelliJIDEA #JAVA

👨‍🏫 **Profesor:** Álvaro García Gutiérrez
📘 **Acceso a Datos**
🗓️ **Clase 4 — 11/11/2025**
🎯 **Tema: Ficheros Binarios en Java | writeUTF / registros fijos / acceso directo**

1️⃣ Introducción

En esta clase seguimos profundizando en **RandomAccessFile**, una herramienta muy potente que permite:

- Leer y escribir datos binarios.
- Controlar la posición exacta del puntero con `seek()`.
- Crear **registros de tamaño variable** (UTF) o **registros de tamaño fijo** (id + nombre, etc.).

El objetivo es entender:

- Cuándo se puede usar **acceso aleatorio real**.
- Por qué `writeUTF/readUTF` **no sirven para acceso directo**, solo secuencial.
- Cómo construir nuestros propios **registros de tamaño fijo** para simular sistemas más cercanos a bases de datos.

La clase se divide en varios bloques:

1. **Ejercicio 3** → Cadenas con `writeUTF` y lectura secuencial.
2. **Ejercicio 4** → Registros fijos (id + nombre fijo) y acceso directo con `seek()`.
3. **Ejercicio 5** →

2️⃣ Ejercicio 3 — Cadenas cortas con writeUTF/readUTF

¿Qué hace writeUTF realmente?

`writeUTF(texto)` NO escribe solo caracteres. Escribe:

1. **2 bytes** → longitud (en bytes) del texto en Modified UTF-8.
2. **N bytes** → contenido codificado en UTF-8.

Esto significa:

- Cada cadena ocupa **tamaño distinto** en el fichero.
- NO existe una fórmula para saltar al “registro 2” directamente.
- La única forma fiable de localizar el siguiente registro es **leer el anterior**.

Por eso el profesor enfatiza:

> “Escribir y leer secuencialmente está bien, pero no se puede hacer acceso aleatorio con UTF.”

2.1 Código completo explicado

```java
package tarea.obligatoria.pkg3.random.acces.file;

import java.io.IOException;
import java.io.RandomAccessFile;

/**
 * Ejercicio 3 — Cadenas cortas con writeUTF/readUTF.
 *
 * writeUTF guarda la longitud de la cadena + sus bytes en formato UTF.
 * readUTF lee el mismo formato y recupera el String tal y como se escribió.
 */
public class Ejercicio3 {

    public static void main(String[] args) throws IOException {

        try (RandomAccessFile raf = new RandomAccessFile("nombres.bin", "rw")) {

            // Al comenzar, dejamos el archivo vacío para evitar errores de restos
anteriores.
            raf.setLength(0);

            // ---- 1) Escritura secuencial de tres nombres ----
            // Aquí podríamos pedirlo por pantalla, pero el profesor los escribe
directamente.
            raf.writeUTF("Ana");
            raf.writeUTF("Luis");
            raf.writeUTF("Marta");

            // El puntero queda al final tras las escrituras.
            // Para leer debemos volver explícitamente al inicio.
            raf.seek(0);

            // ---- 2) Lectura secuencial ----
            String n1 = raf.readUTF();   // Lee longitud + bytes → reconstruye String
            System.out.println(n1);

            String n2 = raf.readUTF();
            System.out.println(n2);

            String n3 = raf.readUTF();
            System.out.println(n3);

            // Mostrar dónde quedó el puntero
            long finalPos = raf.getFilePointer();
            System.out.println("getFilePointer() al final = " + finalPos);

            // ---- Explicación crítica ----
            // NO podemos saltar directamente al segundo nombre con seek(n).
            // writeUTF escribe longitud variable → no existe un tamaño fijo para
calcular posiciones.
        }
```

```
    }
}
```

2.2 Puntos críticos del Ejercicio 3

✔ `setLength(0)`

Vacía el fichero para empezar limpio.

✔ `seek(0)`

Obligatorio para volver a leer desde el principio.

✔ writeUTF = longitud + bytes (tamaño variable)

Cada nombre ocupa espacio distinto ⇒ NO se puede usar acceso aleatorio.

✔ Los datos se leen EXACTAMENTE en el mismo orden en que se escribieron

Si no se mantiene el orden, el puntero queda desincronizado.

3️⃣ Ejercicio 4 — Registros fijos (id + nombre fijo)

Este ejercicio introduce un concepto fundamental:

> Para poder usar acceso aleatorio real con RandomAccessFile, los registros deben tener **tamaño fijo**.

Diseñamos un registro con:

Campo	Tipo	Tamaño
id	int	4 bytes
nombre	10 chars	20 bytes
Total	—	**24 bytes**

3.1 Código completo explicado

```java
package tarea.obligatoria.pkg3.random.acces.file;

import java.io.IOException;
import java.io.RandomAccessFile;

public class Ejercicio4 {

    // Nombre fijo de 10 caracteres → 20 bytes
    static final int NCHARS = 10;

    // Registro fijo: id (4 bytes) + nombre (20 bytes) = 24 bytes
    static final int BYTES_REG = 4 + 2 * NCHARS;

    public static void main(String[] args) throws IOException {
```

```java
        try (RandomAccessFile raf = new RandomAccessFile("personas.dat", "rw")) {

            raf.setLength(0); // Muy importante: limpiamos el archivo

            // ---- 1) Escribir 3 registros de ejemplo ----

            // Registro 0 (offset 0)
            raf.seek(0 * BYTES_REG);
            escribirRegistro(raf, 1, "Ana");

            // Registro 1 (offset 24)
            raf.seek(1 * BYTES_REG);
            escribirRegistro(raf, 2, "Bernardo");

            // Registro 2 (offset 48)
            raf.seek(2 * BYTES_REG);
            escribirRegistro(raf, 3, "Clara");

            // ---- 2) Leer directamente el tercer registro ----
            int indice = 2;
            long offset = indice * BYTES_REG;

            raf.seek(offset);               // Saltamos directo al registro 2
            int id = raf.readInt();         // id = 3
            String nombre = leerNombreFijo(raf, NCHARS);

            System.out.println("Tercer registro -> id=" + id + ", nombre=" + nombre);
        }
    }

    // Escribir registro en la posición actual del puntero
    static void escribirRegistro(RandomAccessFile raf, int id, String nombre) throws
IOException {
        raf.writeInt(id); // 4 bytes
        escribirNombreFijo(raf, nombre, NCHARS);
    }

    // Escribir nombre de longitud fija (rellenando con espacios si hace falta)
    static void escribirNombreFijo(RandomAccessFile raf, String s, int len) throws
IOException {
        if (s == null) s = "";
        if (s.length() > len) {
            s = s.substring(0, len); // recortar si es más largo
        }
        String ajustado = String.format("%-" + len + "s", s); // completar con
espacios
        for (int i = 0; i < len; i++) {
            raf.writeChar(ajustado.charAt(i)); // cada char = 2 bytes
        }
    }

    // Leer un nombre de longitud fija usando un buffer
    static String leerNombreFijo(RandomAccessFile raf, int len) throws IOException {
        StringBuilder sb = new StringBuilder(len); // más eficiente que concatenar
```

```
Strings
        for (int i = 0; i < len; i++) {
            sb.append(raf.readChar()); // readChar lee 2 bytes
        }
        return sb.toString().trim();
    }
}
```

3.2 Puntos críticos del Ejercicio 4

✔ Diseño del registro fijo

- El tamaño de cada registro es **conocido y constante**:
  `BYTES_REG = 24 bytes`.

Esto permite calcular el offset:

```
offset = índice * 24
```

✔ writeChar / readChar → 2 bytes exactos por carácter

Por eso los nombres fijos ocupan:

```
10 chars × 2 bytes = 20 bytes
```

✔ String.format() para rellenar con espacios

`String.format("%-10s", "Ana")` → `"Ana      "`.

✔ Saltar directamente a un registro funciona porque:

- Cada registro ocupa 24 bytes.
- El offset siempre cae en una posición válida.

Ejemplo:

> Registro 2 → 2 × 24 = byte 48

✔ Buffer de lectura (`StringBuilder`)

- Leer todos los chars a un buffer es más eficiente.
- Después se hace `trim()` para quitar los espacios de relleno.

4️⃣ Diferencia conceptual entre Ejercicio 3 y 4

Aspecto	Ejercicio 3	Ejercicio 4
Forma de escribir texto	writeUTF	writeChar (fijo)
Tamaño del registro	Variable	**Fijo**
Se puede calcular el offset	❌ No	✔ Sí
Se puede acceder directamente al registro N	❌ No	✔ Sí
Necesita leer secuencialmente	✔ Sí	❌ No

Aspecto	Ejercicio 3	Ejercicio 4
Uso típico	cadenas simples	estructuras tipo BD

5️⃣ Conclusión de los ejercicios anteriores

- `writeUTF` es potente para almacenar texto, pero NO sirve para acceso aleatorio porque los registros son variables.
- Para acceso directo necesitamos **registros fijos** → misma estructura y mismo tamaño siempre.
- `seek()` permite reposicionar el puntero con precisión, pero solo es útil si sabemos exactamente dónde empieza cada registro.
- `setLength(0)` debe usarse siempre que queramos "resetear" un fichero binario antes de reescribirlo.
- RandomAccessFile es una herramienta básica para modelar estructuras internas de bases de datos simples o ficheros indexados.

📝 Tarea Obligatoria 4 (Ejercicio 5)

Actualización de un campo dentro de un registro fijo en RandomAccessFile

Este ejercicio profundiza en el uso de **RandomAccessFile** para acceder y modificar **solo un campo** dentro de un registro estructurado.

Trabajamos sobre un archivo binario `personas.dat`, donde cada registro tiene un formato **fijo**.

1️⃣ Estructura del registro (muy importante)

Cada persona ocupa exactamente **48 bytes** en el fichero:

Campo	Tipo	Tamaño
id	int	4 bytes
nombre	20 chars	40 bytes (cada char = 2 bytes con writeChar)
edad	int	4 bytes

TOTAL: 48 bytes por registro

Esto nos permite posicionarnos exactamente en cualquier registro usando:

```
offset = índiceRegistro * 48
```

2️⃣ Código completo comentado por bloques

Apertura del fichero y valores constantes

```
static final int NCHARS = 20;                // Tamaño fijo del nombre
static final int BYTES_REG = 4 + 2*NCHARS + 4;  // 48 bytes por registro
```

- `NCHARS = 20` → todos los nombres ocuparán 20 caracteres exactos.

- Como cada `char` ocupa **2 bytes**, el nombre ocupa 40 bytes en total.
- El registro completo: **4 (id) + 40 (nombre) + 4 (edad) = 48 bytes**.

3️⃣ Escritura inicial de registros (solo si el fichero está vacío)

```
if (raf.length() == 0) {
    escribirRegistro(raf, 1, "Ana",   30);
    escribirRegistro(raf, 2, "Luis",  25);
    escribirRegistro(raf, 3, "Maria", 40);
}
```

- Si `personas.dat` está vacío, creamos tres registros básicos.
- Muy útil para no tener que introducirlos manualmente cada vez.
- Cada llamada a `escribirRegistro()` deja el puntero al final del registro escrito.

4️⃣ Solicitud de datos al usuario

```
System.out.print("Introduce ID a actualizar: ");
int idBuscado = sc.nextInt();

System.out.print("Nueva edad: ");
int nuevaEdad = sc.nextInt();
```

Simple lectura del **id a buscar** y la **nueva edad**.

5️⃣ Búsqueda secuencial del registro por id

Aquí está la parte más crítica del ejercicio.

```
long numRegs = raf.length() / BYTES_REG;  // nº total de registros
long regInicio = -1;
```

- Dividimos el tamaño del fichero entre 48 bytes para saber cuántos registros hay.
- `regInicio` almacenará el **offset exacto** donde empieza el registro buscado.

🔍 Recorrido registro por registro

```
for (int i = 0; i < numRegs; i++) {

    long offset = (long) i * BYTES_REG;
    raf.seek(offset);

    int id = raf.readInt();              // Lee solo el id

    if (id == idBuscado) {
        regInicio = offset;
        break;
```

```
    }
}
```

¿Qué ocurre aquí?

- **offset** calcula la posición exacta del registro `i` → `i * 48`.
- `seek(offset)` mueve el puntero al **comienzo del registro**.
- `raf.readInt()` lee solo el id (4 bytes); el nombre y edad NO se leen todavía.
- Si coincide, guardamos la posición y salimos del bucle.

No hace falta saltar manualmente el registro, porque el siguiente `offset` en la siguiente iteración ya recoloca el puntero.

6️⃣ Comprobación de que el id existe

```
if (regInicio == -1) {
    System.out.println("ID no encontrado.");
    return;
}
```

Evita errores si el usuario introduce un id inexistente.

7️⃣ Cálculo del offset exacto del campo "edad"

Esta es la parte clave del ejercicio.

```
long offsetEdad = regInicio + 4 + (2L * NCHARS);
```

Interpretación:

- `regInicio` = inicio del registro
- `+ 4` → saltamos el campo id
- `+ 2 * NCHARS = 2*20 = 40` → saltamos los 20 chars del nombre

Total: regInicio + 44
→ Aquí empieza el campo `edad` (4 bytes).

8️⃣ Escritura de la nueva edad (sin tocar nada más)

```
raf.seek(offsetEdad);   // Posicionar el puntero justo en edad
raf.writeInt(nuevaEdad);
```

Esto sobrescribe únicamente el campo **edad**, respetando:

- id
- nombre

9️⃣ Verificación leyendo el registro completo

```
raf.seek(regInicio);

int id = raf.readInt();
String nombre = leerNombreFijo(raf, NCHARS);
int edad = raf.readInt();
```

- Volvemos al inicio del registro actualizado.
- Leemos los 48 bytes completos.
- Mostramos todo para confirmar el cambio.

🔧 Funciones auxiliares

escribirRegistro()

Escribe un registro completo en la posición actual del puntero:

```
raf.writeInt(id);
escribirNombreFijo(raf, nombre, NCHARS);
raf.writeInt(edad);
```

escribirNombreFijo()

Rellena o recorta a exactamente 20 chars:

- Si el nombre es corto → añade espacios.
- Si es largo → lo recorta.
- Escribe cada char con `writeChar()` (2 bytes).

leerNombreFijo()

Lee exactamente 20 `char` y hace `.trim()` al final.

✔ Conclusión del ejercicio

Este ejercicio demuestra cómo:

- Gestionar **registros de longitud fija** en binario.
- Calcular posiciones exactas dentro del archivo usando **offsets**.
- Actualizar **solo un campo** sin afectar al resto del registro.
- Usar correctamente `seek()`, `writeInt()`, `readInt()` y lectura de cadenas fijas con `writeChar/readChar`.

Es un patrón típico en **ficheros de acceso aleatorio**, muy usado históricamente en bases de datos primitivas y sistemas de almacenamiento estructurado.
